

Configurable simulator for computer architecture and organization

Zaharije Radivojevic^{ID} | Zarko Stanisavljevic^{ID} | Marija Punt

School of Electrical Engineering,
University of Belgrade, Belgrade, Serbia

Correspondence

Zaharije Radivojevic, University of
Belgrade, School of Electrical Engineering,
Bulevar kralja Aleksandra 73,
Belgrade 11000, Serbia.
Email: zaki@etf.bg.ac.rs

Funding information

Ministry of Education, Science, and
Technological Development of the Republic
of Serbia, Grant number: III44009

Abstract

This paper presents a novel visual educational configurable simulator for computer architecture and organization (COCONUT). By using the simulator students can create their own processor with an arbitrary architecture and simple organization on the register transfer level, write an assembly program to be executed on the processor and observe the instruction execution phases of the program on the processor. The students can create their processor by using the simulator of a computer system with fixed and configurable parts. The configurable part allows students to use the simulator for defining the instruction decoding of the processor operation unit and the content of the microprogram memory of the processor control unit. The simulator's usage was evaluated at the University of Belgrade—School of Electrical Engineering on two courses for two consecutive school years. The evaluation results showed that 80% of the students stated that the simulator helped them to better understand the course material and that they had a positive user experience with the simulator.

KEYWORDS

computer architecture and organization, computer engineering education, microprogramming, processor design and implementation, software simulator

1 | INTRODUCTION

Computer architecture and organization is an important area for computer engineering, software engineering and electrical engineering students [9]. For a long period of time there is a recommendation [8,9] issued and constantly updated by respectable authorities in the field of computer engineering, i.e., ACM and IEEE, which states that courses covering topics from computer engineering should be supported by hands-on experience for students.

At the University of Belgrade-School of Electrical Engineering (UB-SEE) there are several courses covering different topics from digital logic design, computer architecture, and organization taught at different modules such as: Computer Engineering, Software Engineering, Electronics, Telecommunications and Information Technologies, Signals and Systems, and Physical Electronics. All of these courses have laboratory exercises that are in the majority of cases

conducted using software simulators, which were developed by the UB-SEE staff. Undergraduate studies are eight semesters long and master studies are two semester long. Each course at UB-SEE is taught for one semester. The first courses where students are introduced to computer architecture and organization fundamentals are Fundamentals of Computer Engineering 2 (FCE2) and Computer Architecture and Organization (CAO). Both FCE2 and CAO courses include topics related to computer organization and computer design. During these courses students are familiarized with important features of computer architecture, logic design of simple processors, and how to implement features of computer architecture on a design of a simple processor.

The major goals of the FCE2 and CAO courses are to familiarize students with computer architecture and the organization of a processor, to allow them to write an assembly program to be executed on the processor, to observe instruction execution phases of the program on the processor

and to be able to create their own processor with an arbitrary architecture and simple organization on the Register Transfer Level (RTL). To cover all the mentioned goals it was decided to use simulators as an alternative to the Integrated Development Environments (IDE). The FCE2 and CAO courses are on the first years of learning hardware related courses where students are already familiar with simulator usage and using the IDE could be too complex due to its wide range of features and characteristics. To familiarize students with different computer architectures, to allow students to write an assembly program and to observe program execution it is possible to use various numbers of simulators specialized for different computer architectures and organizations. Usage of specialized simulators is not possible for the last goal where students are required to create their own processor with an arbitrary architecture and simple organization on the RTL. In order to achieve this goal it is possible to use a simulator that allows the user to create its own processor and to simulate instruction execution. Creation of the processor for the FCE2 and CAO courses is focused on decoding and executing the instructions and modifying the other phases is limited. In order to achieve this teachers can create a complete processor that can be modified by students. This modification could be done both on unstructured (using arbitrary components) and structured (using predefined set of components) manner. Taking into account that the FCE2 and CAO courses are introductory courses on computer architecture structured design manner is preferable and it corresponds to the usage of configurable simulators. As there are many available simulators that cover the topics of computer architecture and organization, the possibility of reusing existing simulators has been analyzed. The analysis showed that none of the analyzed simulators met all our goals and a decision was made to develop a new simulator.

In this paper a new simulator called COCONUT, covering the major goals is presented. This includes details on how the simulator is designed, used, and evaluated. The simulator was designed using the Java programming language as a visual simulator of a computer system using a time driven sequential simulation algorithm. The computer system contains processor with fixed computer organization and configurable computer architecture. The configurable architecture allows students to use the simulator for defining instruction decoding of the processor operation unit and the content of the microprogram memory of the processor control unit. The simulator's usage was evaluated on the FCE2 and CAO courses for the two consecutive school years where 80% of the students stated that the simulator helped them to better understand the course material and had a positive user experience with the simulator.

The paper has the following structure. In section 2 an overview and evaluation of existing simulators for educational purposes in the area of computer architecture and

organization is presented. In section 3 the most important details regarding the implementation and the structure of the COCONUT simulator is given. In section 4 the description of how the simulator could be configured for the project assignment at the FCE2 and CAO courses is described. In section 5 the qualitative and quantitative evaluation of students achievements is given and section 6 concludes the paper.

2 | RELATED WORK

In this chapter, an evaluation of existing simulators related to computer architecture will be presented. The evaluations are conducted according to the goals stated in the Introduction. The simulators that were included in the evaluation were developed after the previous survey [14]. The goals were refined into set of questions whose answers were used to determine whether to develop a new simulator or to use an existing one. The questions were divided into groups related to simulator characteristics and simulator usage. The evaluated simulator characteristics included configurations, architectures, execution levels, and components used. The simulator usage group looks at supported courses, programming language used for its development, origin, and evaluation. The results of the evaluation are presented in Table 1.

Fifteen simulators were considered and their acronyms are given in the first column of the table. In the second column, the name of the University where the simulator was developed is presented. The survey covered simulators from Europe, South and North America, and Asia.

All simulators were developed to be used for undergraduate courses at the first and second year of studies. The students attended different courses: Digital Logic Design (DLD), Computer Architecture (CA), Computer Organization (CO), and Computer Architecture and Organization (CAO). The courses were held at various departments: Computer Science (CS), Information Systems (IS), Computer Engineering (CE), Software Engineering (SE), and Electronic Engineering (EE). Some of the courses were held at more than one department or the simulator was used on more than one course and not only at the University where it was developed. For some of the simulators there were no information about where they were used. The details given in the table were as found in the paper describing the simulator. For each simulator the necessary details were extracted from the corresponding papers that describe that simulator. These details are compiled in Table 1.

Some of the evaluated simulators were developed to be used as desktop applications and some of them as web applications. The majority of the desktop simulators were developed using the Java programming language

TABLE 1 Results of the evaluation

Simulator	Author	Departments and Courses	Programming language	Can simulator's parameters be changed?	Content to be loaded	Types of processor architectures	Level of the simulated system	Types of components used	Evaluation
EduMIPS64 [4,15]	University of Catania, Italy	CA at CE	Java	No	Asm	MIPS64	Functional	N/A	QN
MIPS X-Ray [1,26]	Federal Center for Technological Education of Minas Gerais, Brazil	CAO at CS	Java	No	Asm	MIPS	Functional	N/A	QL
MARIE [11,16]	Mackenzie Presbyterian University, Brazil	CA at CS, IS	Java	No	Asm	Hypothetical machine	Data path	N/A	QN, QL
EDUCache [2,19]	Ss. Cyril and Methodius University, FYRM	CAO at CS	Java	Configurable	Trace	N/A	Functional	N/A	QL
Java based Virtual Lab [23]	SRM University, India	Not specified	Java	No	C	ARM	Functional	N/A	N/A
VSMIS [12]	University of Belgrade, Serbia	CAO at CE, SE	Java	Configurable	Trace	N/A	RTL	N/A	N/A
EASE [22]	University of Northern British Columbia, Canada	CAO at CS	Java	Extendable	Asm	CISC, RISC, URISC	Instruction	N/A	N/A
Visual CPU simulator [7]	Kagawa University, Japan	Not specified	JavaScript	No	Asm	Simple architecture	RTL	N/A	QN
Simple CPU Architecture [20]	Redeemer University College, Canada	CO at CS	Java	No	Raw	Simple 8-bit CPU	N/A	COM SEQ	N/A
Logisim [3]	Saint John's University, USA	DLD at CS, IS, EE, CE	Java	Extendable	Raw	N/A	RTL	COM SEQ HS	QN, QL
SDLDS [24]	University of Belgrade, Serbia	DLD at CE	Java	Configurable	N/A	N/A	N/A	COM SEQ	QN
DEEDS [5]	University of Genoa, Italy	DLD at EE, IS	Delphi	Configurable	Asm	Simplified Z80-CPU	RTL	COM SEQ	N/A
DLD-VISU [21]	Technical University of Darmstadt, Germany	DLD at CS	Java EE	Configurable	N/A	N/A	N/A	COM SEQ SM	QN, QL
ViLLE plug-in [10]	University of Turku, Finland	DLD at IS	JavaScript	Configurable	N/A	N/A	N/A	COM	QL
VSDS [6,13]	University of Belgrade, Serbia	DLD at CE, SE	VisualBasic	Extendable	Raw	Simple RISC and CISC	N/A	COM SEQ HS	QN

(EduMIPS64, MIPS X-Ray, MARIE, EDUCache, Java based Virtual Lab, VSMIS, EASE, Simple CPU Architecture, Logisim, and SDLDS) also one web simulator was developed using Java EE (DLD-VISU). Other desktop simulators were developed using programming languages such as Delphi (DEEDS) and Visual Basic (VSDS) that are more commonly used in business areas rather than in education environments. The information regarding the used programming language and implementation details of the developed simulators can be useful for researchers developing new simulators.

After studying the selected simulators it was noted that some of the simulators allow the user to modify the simulated system to some extent. The extensions can be classified as only configuring the existing system (Configurable) or the ability to create new components (Extendable). Configuring the existing system can include modules, characteristics or values that the user can choose from a limited predefined set. New components can be created using a set of predefined simulator components and then added into the set to be used when creating other components within a hierarchical structure.

The MIPS X-Ray simulator is based on the MARS simulator and does not allow any changes to be made, even though the MARS allows that. The EDUCache simulator is a configurable simulator that allows various cache parameters, such as cache line size and number of blocks, and cache replacement policies, such as FIFO and LRU, to be changed. The VSMIS simulator allows the number of memory modules and addressing modes to be configured. When using the EASE simulator students can be asked to come up with ways of extending the simulator and implement one of their own ideas. The simple CPU Architecture simulator was developed using the Logisim simulator and was not intended to be changed even though Logisim was. The Logisim simulator and SDLDS simulation suite can operate in two different modes. In the first mode it is possible to configure the behavior of the combinational and sequential switching circuits and in the second mode the list of components can be extended. The DEEDS simulation suite is composed of three simulators that cover combinational and sequential logic modules, state machine, and microcomputer interfacing and programming at assembly level where state machine combinational and sequential modules can be drawn. The DLD-VISU simulator can be used to configure behavior of combinational and sequential switching circuits. The ViLLE environment plug-in can be configured using combinational switching circuit. The VSDS simulator can be used to create hierarchical modules that consist of predefined or user defined combinational and sequential switching circuits.

Most of the simulators allow some content to be loaded into the memory and to be interpreted according to the content type. Content that can be loaded is raw content (Raw), assembly programs (Asm) that are interpreted by the

simulator or turned into raw content, high programming language such as C (C) that can be compiled into the assembly program or raw content and execution trace (Trace). Some of the simulators were not intended to be used for simulating processors (N/A).

The evaluated simulators support various types of processor architectures. Most of them support simplified versions of CISC and RISC processors suitable for educational purposes. On the other hand the Java based virtual lab supports the ARM architecture and the EduMIPS64 simulator supports the MIPS64 architecture. Some of the simulators were not intended to be used for simulating computer architecture (N/A).

The simulated systems vary in the level of details that can be observed. The EASE simulator allows only observation of results of instruction execution (Instructional). The MIPS X-Ray, EduMIPS64, EDUCache, and Java based Virtual Lab simulators allow the content of the functional unit to be observed (Functional). The Visual CPU, VSMIS, Logisim, and DEEDS simulators provide observation of the content of the registers at register transfer level (RTL). Both the content of the functional unit and data paths can be observed (Data path) with MARIE simulator. Some of the simulators can be used at the RTL level but were not intended for simulating computer organization (N/A).

Based on the target course the simulators can be divided into two groups. The first group of simulators (Visual CPU simulator, EASE, VSMIS, Java based Virtual Lab, EDUCache, MARIE, MIPS X-Ray, EduMIPS64) are those intended for simulating processor architecture and organization and their main purpose is not logic design (N/A). Those simulators are executing instructions for already existing computer architectures. Programs written in assembly language or machine code can be loaded into these simulators. The program instructions can then be executed and the program state can be observed.

The second group of simulators is intended to be used for digital logic design. The simulated systems use various types of modules that include: combinational modules (COM), sequential modules (SEQ), state machines (SM), and modules with hierarchical structure (HS). Those simulators can use either simple components like combinational and sequential switches or modules. Some of them are meant for testing students in the design of simple digital modules (DLD-VISU, ViLLE plug-in), while others can be used to create simple modules using state machines or Karnaugh maps (SDLDS, DEEDS, DLD-VISU) or more complex modules with a hierarchical structure (Logisim, VSDS).

The majority of observed simulators have been evaluated from the following aspects: evaluation of students' achievements that used the simulators, evaluation of the simulator's usability and evaluations of the user experience. The student's achievements were evaluated through quantitative

measurements of success such as grades, points, time, and number of actions needed for task completion (Quantitative-QN). The simulator's usability and user experiences were evaluated through different types of self-reported data gathered through various types of questioners such as Likert scales, semantic scales, and open-ended questions (Qualitative-QL). The authors of some papers did not mention any evaluation in the description of their simulators (N/A). The evaluation of different simulators was conducted in a non-uniform manner depending on multiple factors including University policies, development goals, and previous user experience.

After careful evaluation of the simulators in the context of the goals presented at the start of the paper the simulators that were meant only for computer architecture and organization courses did not meet the goal of supporting multiple architectures. The simulators that were meant for digital logic design courses had no clear distinction between computer architecture and computer organization. Based on these results the decision was made to develop a new computer system simulator on the RTL level. Parts of the computer organization that are the same across computer variations are referred to as fixed computer organization and parts of the computer organization that are strongly influenced by variations in the computer architecture are referred as configurable computer architecture. The simulator will have fixed computer organization and configurable computer architecture in order to allow students to observe the connection between computer architecture and computer organization.

The proposed system to be simulated will consist of:

- Fixed computer organization: three internal buses, one ALU unit, one ADD unit, a control unit with horizontal microprogramming, a fixed number of input-output devices, and a fixed level of interrupts.
- Configurable computer architecture: number of registers, data size, addressable units, an instruction decode unit that is fully configurable, a fully configurable control unit (microprogramming memory and microprogram jumps), support for loading raw memory content.

The proposed simulator will consist of:

- Logic: combinational, sequential modules, hierarchical modules, and configurable modules.
- Execution: loading the program to be executed, executing the program (clock level, instruction level, and program level).
- Presentation: observing signals (current value and color coding), observing current state (table presentation), and

observing historical data (states and values using time diagrams).

3 | COCONUT IMPLEMENTATION

The simulator was implemented using the Java programming language version 1.8. During implementation the SLEEP methodology was considered [17,18]. According to the SLEEP methodology educational simulators intended for computer architecture and organization courses could consist of five layers: simulation, logic, execution, presentation, and physics. The simulation layer enables the simulation execution in a distributed environment. The logic layer describes the logical behavior of simulated components and their connections. The execution layer calculates the state of the simulated components and simulation time propagation. The presentation layer visualizes the structure and the state of the simulated components. The physics layer describes the physical behavior of the simulated components and their connections. Considering the goals of the simulated system logic, both the execution and presentation layers were used as presented in Figure 1.

The Logic layer consists of a set of classes and interfaces that describe the structure, behavior, and connection between components. The component's structure and behavior are represented by the *LogicComponent* interface. The connections between components are represented by the *Pin* class.

The *LogicComponent* interface enables access to the logic component's inner structure and behavior. The inner structure consists of inputs, outputs, states, and subcomponents that creates hierarchical components. The Logic component can be defined using a programming language or using other components, pins and their connections. The programming language was used when defining standard combinational modules, sequential modules, and configurable components. The user can define hierarchical components using other components that are either programmable or hierarchical such as processor, memory, and input/output blocks. The behavior of each logic component defines how state and outputs of the component are changed depending on a change of input and elapsed simulation time. Logic components usually have a single graphical representation.

The *Pin* class enables access to its inner state and a list of the connected logic components and their pins. The *Pin*'s state changes depending on the behavior of the component it is attached to and unmodified change is propagated to connected components with zero delay. The *Pin*'s graphical representation can consist of one or more lines that does not have to be graphically connected.

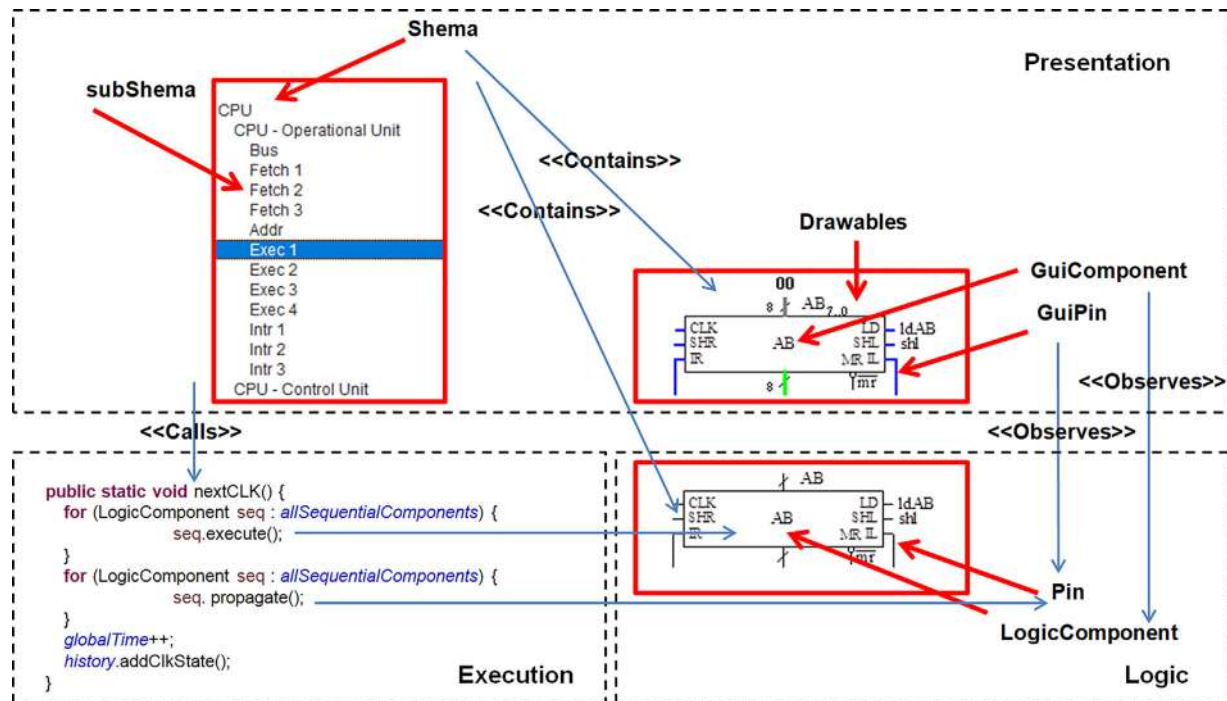


FIGURE 1 Layers of the simulator and their connections

The initialization of logic components and pins is performed in three steps. The creation of all logic components and their output pins is performed in the first step. In the second step the connections between components are created, an output pin of a component is set to be an input pin of a connected component. At the end of initialization, in the third step, logic components and pins are connected with their graphical representations in the Presentation layer.

Based on the characteristics of the system that is simulated and the user requirements in terms of speed and execution time different simulation algorithms can be used. The goal of the simulator was the simulation of the CPU at the RTL level for educational purposes. Considering the level of complexity of CPUs used in the mentioned courses execution was performed using a sequential simulation algorithm. There was no need for parallel or distributed simulations that are used for complex and time consuming simulations.

Sequential simulation algorithms can be classified into two basic categories: time driven and event driven. When performing the execution loop in a time driven algorithm each component is evaluated and their result is scheduled for the next execution phase. A positive aspect of this algorithm is its simplicity, but the drawback is the time needed to evaluate each component in every iteration. During the execution loop in an event driven algorithm events are taken from the event queue and executed on the specified component. The execution can create new events that are subsequently inserted into the event queue and afterwards the event queue is sorted. In case where there are not too many events to be

executed this algorithm can be faster than the time driven one. Otherwise, if there are too many events scheduled for the same moment this algorithm has degraded performance. The COCONUT simulator was implemented using the time driven sequential simulation algorithm.

Time driven algorithms evaluate each component in every iteration propagating for a minimal quant of time (controlled propagation). In digital systems consisting of sequential and combinational modules, modeled with zero delay, the algorithm can be accelerated. The algorithm first evaluates sequential modules and then propagates changes to the connected combinational modules that are further propagating the change (uncontrolled propagation). The simulator is implemented using time driven sequential simulation algorithm with uncontrolled propagation having the single clock as shown in Figure 1.

The method *nextCLK* represents one iteration of the time driven algorithm propagating for one clock. When the user chooses the moment for observing the simulation state this method is called one or multiple times. The user can select whether to go to the next clock, next instruction, to the end of program or to an arbitrary time that can be either before or after the current simulation time. The *nextCLK* method consists of two phases: the new state calculation and the propagation of the new state to the connected modules. After these two phases are performed historical data are stored to be used in the Presentation layer and potentially for rollbacks.

In the first phase the *execute* method is called for each sequential component. This method evaluates sequential module inputs and the current state, calculates the new state of

the module and calculates the values for the module's outputs. The inputs used for evaluation were calculated in the previous iteration of the simulation loop. The first phase ends when all sequential modules are evaluated and the next phase of the algorithm can start.

In the second phase the method propagate is called for each sequential component. This method propagates the output pin's value of the selected sequential component to the input pins of connected components. In case when the connected component is a sequential module the propagation stops, otherwise when the connected component is a combinational module the evaluation is performed and followed by the propagation. During evaluation of the combinational module all input pins' values are processed and the new values for the module's output pins are calculated. If this new value differs from the previous value the propagation continues to the next connected components. In this way (changes only propagation) the execution of the algorithm is accelerated. Propagation is done using a depth first strategy. Detection of unstable infinite combinational loops is performed dynamically during simulation and if the loop occurs the user is notified.

After the second phase the simulation time and historical data are updated. The simulation time is incremented which corresponds to a single clock tick. The same algorithm could be applied to a system consisting of multiple clocks and in that case the increment period would correspond to the least common multiple of the clocks. Historical data contains the states of all sequential modules and the values of all output pins attached to logic components whether it is a combinational or a sequential component. This data could be used for displaying time diagrams and for rollbacks.

Before the first call of the simulation execution loop (*nextCLK*) the initialization of all combinational and sequential modules is performed (*init* method). The *init* method is also called during simulation each time when a user changes the state of some sequential module. The *init* method

is similar to the *nextCLK* method, however the *nextCLK* method is applied only to the sequential modules and the *init* method to both sequential and combinational modules.

The Presentation layer is intended to visualize the structure and state of the simulation logic layer. In order to model the system components interface the *Schema* interface is used. This interface contains methods for extracting inner logic component (*LogicComponent*), for visualization of inner logic component (*GuiComponent*), for obtaining subschemas (*Schemas*) and for getting the graphical interface (*Drawables*) when used as a subschema. Each *Schema* contains one *LogicComponent* that describes the *Schema's* behavior when simulated. The *GuiComponent* class describes the *Schema's* presentation. It contains a number of graphical elements (*Drawable*) that can be linked to the state and values of corresponding Logic layer components and it contains its position on the user interface. Classes that implement the *Schema* interface can be simple or hierarchical whether they describe one or multiple other schemas forming a Composite pattern. The *Drawable* elements describe *Schema's* user interface when used within other components. The *Schema* interface also contains methods for initialization of logic component, logic connections, and graphical interface.

When presenting the state of a logic component either the current state or historical data containing previous states can be shown. The current state can be presented either graphically or by using tables. Time diagrams are used for presenting historical data. All three types of presentation are dynamically examining the lists of sequential and combinational components and their pins in order to display their values.

The graphical presentation of the current state of a logic component uses series of connected straight lines and labels. Depending on the type and the state of a logic component the lines can be presented using different color codes and line weight. The state of the component can be a logic one, a logic zero, a multi-value, or a high zero. Those states correspond to the blue, red, gray, and green color codes respectively. Line

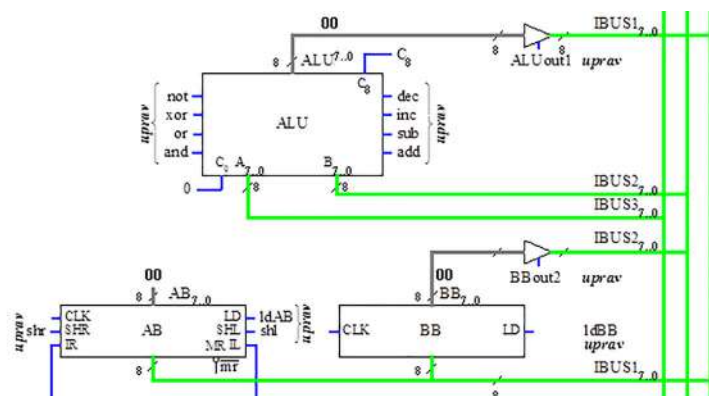
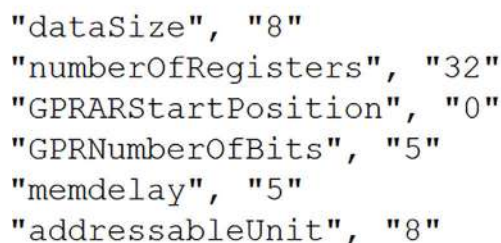


FIGURE 2 Configuration of general parameters (influence of data size to execution unit)

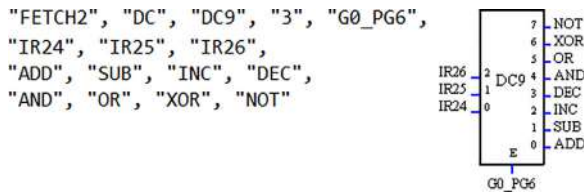


FIGURE 3 Configuration of a decoder used for instruction decoding and addressing mode decoding (arithmetic and logic instruction decoding)

weight depends on the number of pins linked to the line. Labels correspond to the current state of a logic component and display values in either hexadecimal or binary notation.

Table presentation of the current state of sequential components is used for registers, flip-flops, and memory modules. The current state can be presented in hexadecimal or binary notation. During simulation the user can interact with the table presentation to dynamically modify the current state of the sequential component which corresponds to the first phase of the simulation algorithm. This modification will affect all connected combinational modules in the same manner as the second phase of the simulation algorithm.

Time diagrams are used for observing historical data of sequential components' states and combinational components' values. The time diagram consists of multiple signals whose values can be presented as a logic one, a logic zero, a multi-value, or a high zero. The set of signals to be displayed, its order and time interval can be dynamically changed by the user.

4 | USAGE OF THE COCONUT

The COCONUT simulator was used as an educational tool on a project assignment for both FCE2 and CAO courses. The goal of the project assignment was to help students to create and test their own processor for the given architecture and organization on the RTL level, to allow them to write assembly programs to be executed on the processor and to observe instruction execution phases of the program on the processor.

By using the simulator the students can create their own processor with a given computer architecture and computer organization. The computer organization consists of both fixed and configurable parts. The fixed parts include three internal buses, one ALU unit, one ADD unit, a control unit

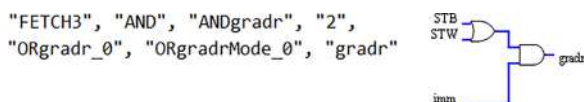


FIGURE 4 Configuration of logic circuits signals used for generating operation unit signals (addressing mode error signal)

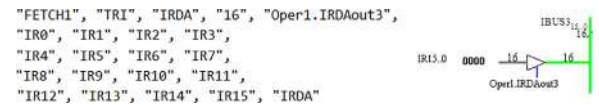


FIGURE 5 Configuration of tri state buffers used to connect registers to busses (instruction register connected to bus 3)

with horizontal microprogramming, a fixed number of input-output devices, and a fixed level of interrupts. The configurable parts include the number of registers, data size, addressable units, an instruction decode unit, and control unit. The configuration of the simulator can be done using two files (configuration.txt and microprogram.txt). The configuration.txt file is used to create structural schemas of both the operation and control unit. The microprogram.txt file is used to initialize the microprogram memory in the control unit.

The parameters that can be changed in the simulator through the configuration.txt file can be divided into three categories: general parameters, operation unit parameters, and control unit parameters. The general parameters include data size, the number of general purpose registers, the number of bits used for addressing general purpose registers, memory delay, and addressable unit size (Figure 2).

The operation unit parameters include decoders used for instruction and addressing mode decoding (Figure 3), logic circuits used for generating operation unit signals (Figure 4), tri state buffers used to connect registers to busses (Figure 5), and logic circuits used for generating instruction status (Figure 6).

The control unit parameters include combinational modules for generating addresses for multiple conditional branches (Figure 7) and combinational module for generating control unit branch signals (Figure 8).

The content of the microprogram memory can be set in the simulator through the microprogram.txt file. The control unit is implemented using horizontal microprogramming with an instruction format that includes both the operation and control part in the same microinstruction. By editing the microprogram.txt file a user can define the content of the microprogram memory while its internal structure and meaning of

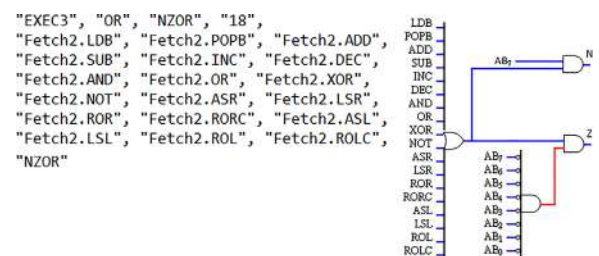


FIGURE 6 Configuration of logic circuits used for generating instruction status Operation (N and Z indicators)

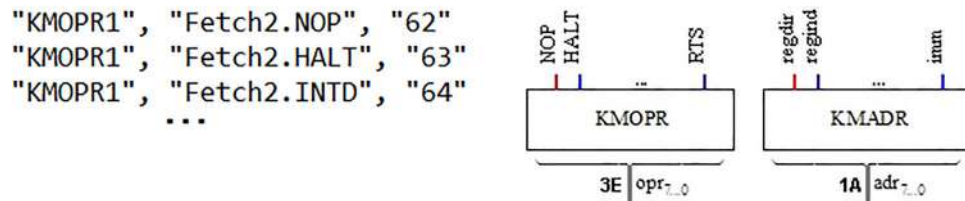


FIGURE 7 Configuration of combinational modules for generating addresses for multiple conditional branches (instructions and addressing modes)

certain bits of operation and control part of the microinstruction is defined using the configuration.txt file. Examples of microinstructions are presented in Figure 9.

Using the simulator students can test the created processor. After writing the configuration.txt and microprogram.txt files the user can start the simulation of the processor. In order to test the created processor the user can write machine code and load it into memory as raw content. The main window of the simulator is presented in Figure 10. The structural tree (ST) shows the hierarchy of the simulated system and allows the user to choose which functional unit will be presented on the central panel. The central panel (CP) is used for visual presentation of

combinational and sequential modules of the selected functional unit of the simulated system. The visual presentation includes the component's layout, connections, states, and values. The watch list (WL) allows the user to open the dialog window in which the current state of the memory, flip-flops, and registers can be observed. The memory, flip-flops, and registers values can be changed dynamically during simulation using the opened dialog widows in the status area (SA). The execution area (EA) enables user to run the simulation for one clock or instruction in advance, up to a given clock value or run the entire program until completion. The control unit status area (CU) shows the current microinstruction that is executed and trace of all microinstructions that have been

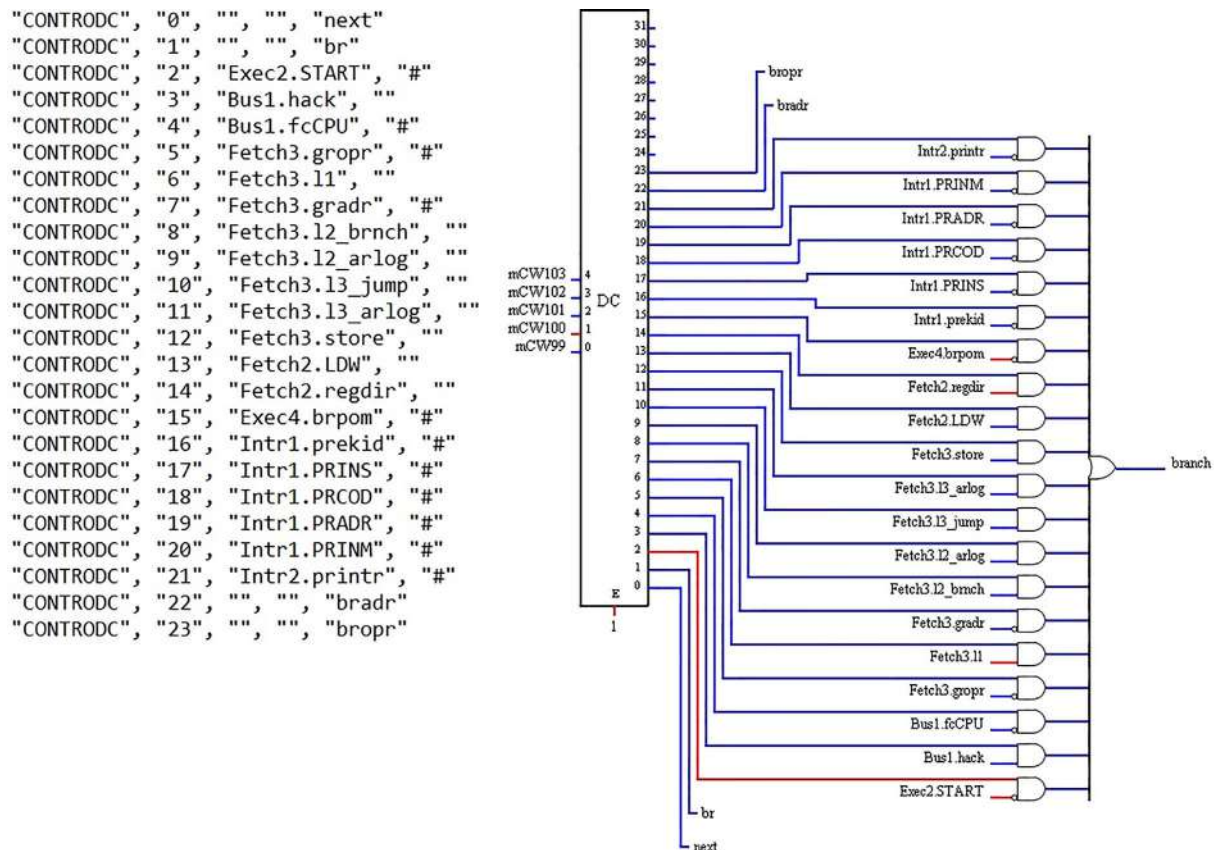


FIGURE 8 Configuration of combinational module for generating control unit branch signals (24 branches from which 20 are conditional branches and 4 are unconditional branches)

```

madr01 PCout1, MOST1_2, ldMAR, incPC;
(a)
madr05 br (if !gopr then madr07);
(b)
madr0A eMAR, rdCPU, br (if !fcCPU then madr0A);
(c)
madr19 bradr
(d)

```

FIGURE 9 Microinstruction examples with a) operation part only, b) control part only, c) both operation and control part, d) multiple conditional branch

executed since the simulation was started. The configuration area (CA) consists of multiple tabs and allows the user to edit the configuration and microprogram, to observe errors and time diagrams.

5 | RESULTS

The COCONUT simulator has been used at the FCE2 course since the 2015/2016 school year and at the CAO course since the 2014/2015 school year. FCE2 a mandatory course in the 3rd (out of 8) semester for the Computer Engineering and the Software Engineering modules and CAO course is a mandatory course in the 5th (out of 8) semester for the Electronics module. The final grade on these courses depends on both practical and written assignments. The practical aspect includes a project assignment (20%) and mandatory laboratory exercises (20%). The rest of the grade (60%) is obtained through a midterm and final exam. The experiences from using the

COCONUT simulator at these two courses were conducted using quantitative and qualitative evaluation.

In order to test how the introduction of the COCONUT simulator influenced the courses the following parameters were collected:

- P1.** Total number of students that took the exam in all exam terms during the school year
- P2.** Total number and percentage of students who passed the project part (defended their project)
- P3.** The average number of points (out of 80) at all other course activities except the project part for students who passed the project part
- P4.** The average number of points (out of 80) at all other course activities except the project part for students who did not pass the project part

All these parameters were collected for two consecutive school years (Y1 and Y2) in which the COCONUT simulator was used and also for one school year before the introduction at each course (Y0). For students who took the exam more than once during a school year, only the last score was used. Table 2. shows statistics for both courses. It is interesting to notice that for the first school year (Y1) students could chose weather to use simulator or to submit a project as in the previous year (Y0) and only one student out of 171 chose to submit the project as in the previous year.

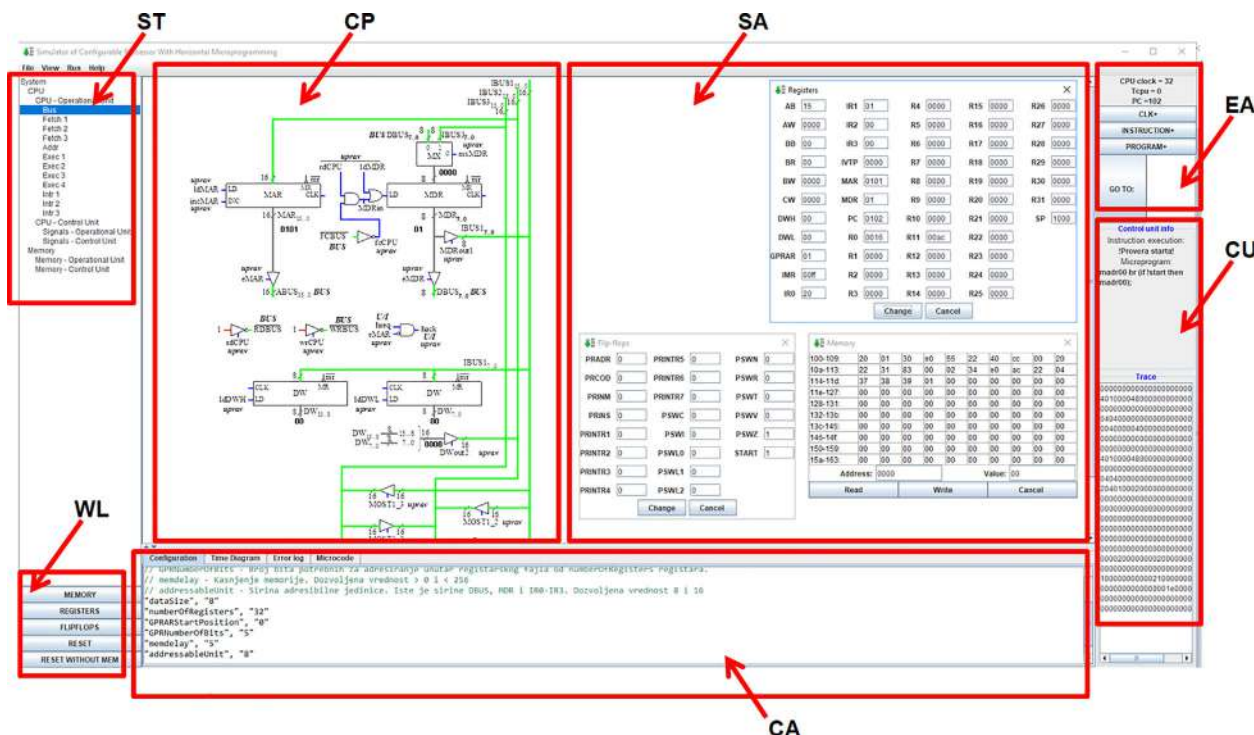


FIGURE 10 The COCONUT simulator main window

TABLE 2 Statistics for the FCE2 and CAO courses

	Y0	Y1	Y2
P1	273	331	326
P2	155 (57%)	171 (52%)	179 (55%)
P3	67	68	67
P4	58	54	55

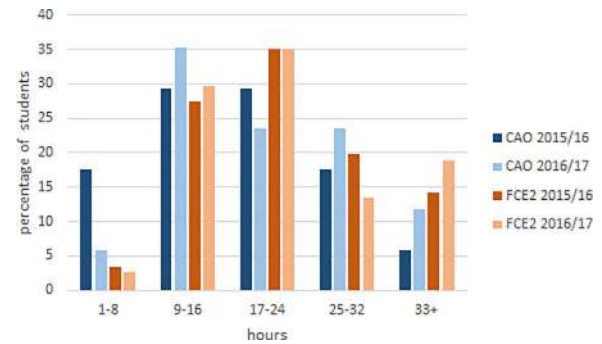
An analysis of collected results showed that after the introduction of the COCONUT simulator none of the P2, P3, and P4 parameters changed significantly. These results could derive from the fact that for both the FCE2 and CAO courses most of the students chose to defend the project assignment after the final exam. In order to confirm the conclusion of the QN analysis the two-way ANOVA statistical test was used. The first factor in the analysis was the school year (Y0, Y1, Y2) and the second factor was the group of students (students who defended the project, students who did not defend the project). As the dependent variable students' average number of points (out of 80) at all other course activities except the project part was used. The test showed that the school year factor is not significant $F(2,927) = 1.77, p = .1709$, the group of students factor is significant $F(1,927) = 212, p < .0001$, and the interaction between the two factors is not statistically significant $F(2,927) = 1.52, p = .2193$.

In order to obtain students' feedback a survey was conducted on students who successfully completed the project. The survey was conducted on both courses in two consecutive school years (2015/16 and 2016/17) and was taken by 162 students in total, 91 and 37 students at the FCE2 course, respectively, and 17 students at the CAO course at each school year. The survey consisted of interval scale, Likert scale, and open-ended questions as suggested in [25].

The interval scale question (IS1) was used to find out how much time the students needed to finish their project. The answers were provided in intervals of 8 hr (one working day). Figure 11. shows the obtained results for FCE2 and CAO for the 2015/16 and 2016/17 school years. For both courses at both school years more than 80% of students completed the project in not more than 4 days. The results showed that the difficulty of the project was well estimated.

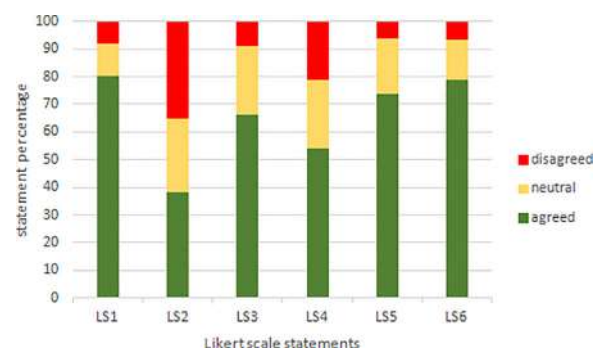
The Likert scale statements were:

- LS1.** The simulator helped me to better understand the course material.
- LS2.** The project assignment should be a mandatory prerequisite for the final exam.
- LS3.** The simulator has a user friendly interface.
- LS4.** The simulator's configuration process was easy.
- LS5.** The simulator's visual representation allows the user to easily follow the details of the instruction execution.
- LS6.** Overall user experience with the simulator was positive.

**FIGURE 11** Number of hours needed for the project completion

The scale items were: strongly disagree, disagree, neither agree nor disagree, agree, and strongly agree. Likert scale results were gathered and analyzed by extracting the percentage of students who indicated agreement with the statements by using the top two answers (agree and strongly agree) for each of the questions. The obtained overall percentages for both school years and on both courses are shown in Figure 12. On the x-axis the six different statements are presented, and on the y-axis the percentage of students that agreed (agree and strongly agree items), were neutral (neither agree nor disagree item) and disagreed (disagree and strongly disagree items) with the Likert scale statements are given.

The obtained results showed that simulator helped for approximately 80% of students to better understand the course material (LS1) and that they were satisfied with the overall user experience with the simulator (LS6). The students preferred to be able to choose whether to defend the project assignment before or after the final exam in comparison to having the project assignment as a mandatory prerequisite for the final exam (LS2). The three questions regarding the simulator's usability showed that students were the least satisfied with the simulator's configuration process using txt files (LS4) and that they were most satisfied with the simulator's visual representation that allowed them to easily follow details of the instruction execution (LS5). Overall they found simulator to have a friendly user interface (LS3).

**FIGURE 12** Percentage of students that agreed, disagreed or were neutral with Likert scale statements

The open-ended questions were:

- OE1.** Should anything be changed in the simulator and if yes what?
- OE2.** Should anything be changed in the configuration process of the simulator and if yes what?
- OE3.** Should anything be changed in the way of examination for the practical project part and if yes what?
- OE4.** Should the practical project examination be held during the semester before the exam terms?
- OE5.** Should a laboratory exercise be added with the use of the simulator as a preparation for the practical part?
- OE6.** Name biggest pros and cons in working with the simulator.

For the OE1 question the electronics department students pointed at some bugs (e.g., full screen view), while the computer engineering and software engineering students had a lot of suggestions for improving the design and functionalities. For the OE2 question all students agreed that some sort of editor should be provided instead of the txt files for configuration. For the OE3 and OE4 questions the majority of the students thought that nothing should change. For the OE5 question most of the students thought that it would be a good idea to introduce such a laboratory exercise. When it comes to the biggest cons the students mostly repeated their answers from the previous questions, and when it comes to pros they agreed that it helped them to understand how a processor works using practical examples (OE6).

6 | CONCLUSION

In this paper the COCONUT simulator was presented. The main goal of this simulator was to allow students to observe the connection between computer architecture and computer organization by creating and testing their own processor for a given architecture and organization on the RTL level. The students can create their processor by using the simulator of the computer system with fixed and configurable parts. The configurable parts of the system include how instructions are decoded in the processor operation unit and what is the content of the microprogram memory of the processor control unit. In order to test the created processor the students have to write their own program, load it into memory, and observe the program execution. The simulation of the processor that executes the program can be run for one clock or one instruction in advance, up to a given clock value, or until the program's conclusion.

The COCONUT simulator has been used for the two consecutive years at two courses at the University of Belgrade—School of Electrical Engineering. The experiences of using the COCONUT simulator at these two courses were obtained using quantitative and qualitative evaluation. Results of the

quantitative evaluation are consistent with the student's achievements before introducing the simulator and showed that students that finished the project assignment achieved a significantly better final grade and scored better on laboratory exercises, midterm, and final exam. The qualitative evaluation was performed by collecting self-reported data using surveys with interval scale, Likert scale, and open-ended questions. The results of the interval scale question showed that more than 80% of the students completed the project in not more than four days. The data collected from the Likert scale questions showed that 80% of the students stated that the simulator helped them to better understand the course material and had positive user experience with the simulator. The observations gained from the open-ended questions were used to constantly improve the simulator over the last two years.

Future work could include extensions to the existing system functionality and the addition of new functionality to the system. Possible extensions could include different modes for configuring logic circuits used for generating operation unit signals. Instead of single gate configuration using textual files, support for Boolean expressions or graphical editing could be added. New functionality could include a configurable assembler, a configurable operation unit with different organization such as various numbers of buses and configurable control unit with different type of organization such as vertical microprogramming and hardwired organization.

ACKNOWLEDGEMENT

This work was supported by the Ministry of Education, Science, and Technological Development of the Republic of Serbia [III44009].

ORCID

Zaharije Radivojevic  <http://orcid.org/0000-0002-9412-7699>

Zarko Stanisavljevic  <http://orcid.org/0000-0003-0272-5139>

REFERENCES

1. M. R. D. Araujo et al., *MIPS X-Ray: A MARS simulator plug-in for teaching computer architecture*, Int. J. Recent Contr. Eng. Sci. IT **2** (2014), 36–42.
2. B. Atanasovski et al., *EDUCache simulator for teaching computer architecture and organization*, IEEE Global Engineering Education Conference (EDUCON), **2013**, 1015–1022.
3. C. Burch, *Logisim: A graphical system for logic circuit design and simulation*, J. Edu. Resources Comput. **2** (2002), 5–16.
4. V. Catania et al., *An open and platform-independent instruction-set simulator for teaching computer architecture*, WSEAS Trans. Info. Sci. Appl. **11** (2014), 42–50.

5. G. Donzellini and D. Ponta, *A simulation environment for e-learning in digital design*, IEEE Trans. Ind. Electron. **54** (2007), 3078–3085.
6. N. Grbanovic, B. Nikolic, and J. Djordjevic, *The VSDS environment based laboratory in computer architecture and organisation*, Comp. Appl. Eng. Educ. **19** (2011), 685–696.
7. S. Hara et al., Design, implementation and trial evaluation of CPU simulator to visualize register-transfer level micro-operation, The Third International Conference on Electronics and Software Science (ICESS2017), (2017), 7–12.
8. IEEE Computer Society and ACM, *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*, 2004. [Online]. Available online at: <https://www.acm.org/binaries/content/assets/education/curricula-recommendations/ce-final-report.pdf> [Accessed January 2018].
9. IEEE Computer Society and ACM, *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*, 2016. [Online]. Available online at: <http://www.acm.org/binaries/content/assets/education/ce2016-final-report.pdf> [Accessed January 2018].
10. V. Karavirta et al., *Interactive Exercises for Teaching Logic Circuits*, The 2016 ACM Conference on Innovation and Technology in Computer Science Education, (2016), 101–105.
11. T. Kurihara, J. F. M. Soares, and L. T. M. Raunheite, *The use of MARIE® CPU simulator in the Computer Architecture (CA) Course: An exploratory analysis to assess the Students' perception of learning*, J. Lit. Art Studies **6** (2016), 1434–1444.
12. K. Milenković, Ž. Stanisavljević, and J. Đorđević, *Visual software system for memory interleaving simulation*, Serb. J. Elec. Eng. **14** (2017), 51–66.
13. B. Nikolić, N. Grbanović, and J. Đorđević, *The visual simulators for architecture and computer organization learning*, J. Automatic Control **19** (2009), 31–34.
14. B. Nikolic et al., *A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization*, IEEE Trans. Edu. **52** (2009), 449–458.
15. D. Patti et al., *Supporting undergraduate computer architecture students using a visual mips64 cpu simulator*, IEEE Trans. Edu. **55** (2012), 406–411.
16. P. W. C. Prasad et al., *Using simulators for teaching computer organization and architecture*, Comp. Appl. Eng. Educ. **24** (2016), 215–224.
17. Z. Radivojevic, M. Cvetanovic, and J. Djordjevic, *Design of the simulator for teaching computer architecture and organization*, IEEE 2nd Eastern European Regional Conference on the Engineering of Computer Based Systems (ECBS-EERC), (2011), 124–130.
18. Z. Radivojević, M. Cvetanović, and Z. Jovanović, *Reengineering the SLEEP simulator in a concurrent and distributed programming course*, Comp. Appl. Eng. Educ. **22** (2014), 39–51.
19. S. Ristov et al., *Using EDUCache simulator for the computer architecture and organization course*, Int. J. Eng. Pedagogy **3** (2013), 47–56.
20. D. C. Schuurman, *Step-by-step design and simulation of a simple CPU architecture*, The 44th ACM technical symposium on Computer science education, (2013), 335–340.
21. A. Shoufan, Z. Lu, and S. A. Huss, *A web-based visualization and animation platform for digital logic design*, IEEE Trans. Learn. Technol. **8** (2015), 225–239.
22. N. Skillen, V. Manickam, and A. Aravind, *EASE: an extensible architecture simulation engine*, ACM 16th Western Canadian Conference on Computing Education, (2011), 23–27.
23. M. K. Srilekha, P. Ponnammal, and R. Bhakkiyalakshmi, *JAVA based virtual lab for embedded processor design*, Indian J. Sci. Technol. **9** (2016), 1–4.
24. Z. Stanisavljevic et al., *SDLDS—System for digital logic design and simulation*, IEEE Trans. Edu. **56** (2013), 235–245.
25. T. Tullis and B. Albert, *Measuring the user experience*, 2nd ed., Morgan Kaufman, Burlington, MA, 2008.
26. K. Vollmar and P. Sanderson, *MARS: an education-oriented MIPS assembly language simulator*, ACM SIGCSE Bulletin **38** (2006), 239–243.



Z. RADIVOJEVIC received his BSc (2002), MSc (2006), and PhD (2012) degrees in Computer Engineering from University of Belgrade, School of Electrical Engineering. He is currently an associate professor at the University of Belgrade, School of Electrical Engineering at the

Department of Computer Engineering and Information Theory, teaching several courses on computer architecture and organization, e-business infrastructure, and mobile device programming. His research interests include computer architecture and organization, concurrent and distributed programming, data analysis, simulations, and reverse engineering.



Z. STANISAVLJEVIC received his BSc (2007), MSc (2008), and PhD (2015) degrees in Computer Engineering from University of Belgrade, School of Electrical Engineering. He is currently an assistant professor at the University of Belgrade, School of Electrical Engineering at the Department of Computer Engineering and Information Theory,

teaching several courses on computer architecture and organization and computer security. His research interests include eLearning tools, computer architecture and organization, network security, cryptography, and internet programming.



M. PUNT received her BSc (2004), MSc (2009), and PhD (2015) degrees in Computer Engineering from University of Belgrade, School of Electrical Engineering. She is currently an assistant professor at the University of Belgrade, School of Electrical Engineering at the

Department of Computer Engineering and Information Theory, teaching several courses on computer architecture and organization, web design, and human–computer interaction. Her research interests include computer architecture, digital systems simulation, consumer electronics, and human–computer interaction.

How to cite this article: Radivojevic Z, Stanisavljevic Z, Punt M. Configurable simulator for computer architecture and organization. *Comput Appl Eng Educ*. 2018;26:1711–1724. <https://doi.org/10.1002/cae.22034>